Name:

Dr.-Ing. Hartmut Helmke
Fachhochschule
Braunschweig/Wolfenbüttel
Fachbereich Informatik

Matrikelnummer:	Punktzahl:
Ergebnis:	

Freiversuch ☐ F1 ☐ F2 ☐ F3 ☐

Klausur im WS 2003/04 :

Informatik III

Hilfsmittel sind bis auf Computer, Handy etc. erlaubt !	Bitte Aufgabenblätter mit abgeben !
Austausch von Hilfsmitteln mit Kommilitonen ist **nicht** erlaubt !	

Die Lösungen können in einigen Fällen hier auf dem Aufgabenblatt angegeben werden. Sie dürfen aber auch Ihre Lösungen, falls erforderlich, auf separaten Aufgabenblättern notieren.
Bitte notieren Sie auf **allen** Aufgabenblättern und separaten Blättern Ihren Namen bzw. Ihre Matrikelnummer. Auf eine absolut korrekte Anzahl der Blanks und Zeilenumbrüche braucht bei der Ausgabe nicht geachtet zu werden. Dafür werden keine Punkte abgezogen.
Hinweis: In den folgenden Programmen wird sehr häufig die globale Variable *datei* verwendet. Hierfür kann der Einfachheit halber die Variable *cout* angenommen werden. Die Variable *datei* diente lediglich bei der Klausurerstellung dem Zweck der Ausgabeumlenkung.

Geplante Punktevergabe

Punktziel	Sonderpunkte	erreicht
A1: 37 P.		
A2: 28 P.		
A3: 11 P.		
A4: 9 P.		
A5: 15 P.		
Summe 100 P.		

Aufgabe 1 : Methodenimplementierung

ca. 42 Punkte
Gegeben ist die Klasse `Mitarbeiter`.

Listing 1: (./Code1/Mitarbeiter.h)

```
#include <fstream>
#include <string>
using namespace std;

// globale Variable fuer Ausgabedatei
// kann entsprechend cout verwendet werden
extern ofstream datei;

class Mitarbeiter  {
   protected:
      string name;
      double gehalt;
      Mitarbeiter* p_chef;
   public:
      Mitarbeiter(double g=3.0,string n="A");
      ~Mitarbeiter();
      virtual double GetGehalt() {return gehalt;}

      // Aufgabe a)
      // g wird bei gehalt eingetragen
      void SetGehalt(double g);
      // Es werden die anz Elemente aus dem
      // Feld t addiert und gehalt zugewiesen
      void SetGehalt(double t[], int anz);
      // Rueckgabe von name
      string GetName();
      // n in name eintragen
      void SetName(string& n);
      // Inhalt von pn in name eintragen
      void SetName(string* pn);
      // pc bei p_chef eintragen
      void SetChef(Mitarbeiter* pc);
      // c entsprechend bei p_chef setzen
      void SetChef(Mitarbeiter& c);
   };

void TestKompliziert();
```

a.) Implementieren Sie die sieben Methoden `SetGehalt` ... `SetChef` an der dafür vorgesehenen Stelle in der Quellcode-Datei `Mitarbeiter.cpp` auf einem Extrablatt.
b.) Zu welcher Ausgabe in `datei` führt der Aufruf von der Funktion `TestKompliziert`

Listing 2: (./Code1/Mitarbeiter.cpp)

```
#include "Mitarbeiter.h"


Mitarbeiter::Mitarbeiter(double g, string n):
name(n), gehalt(g), p_chef(0) {
   datei << "+M␣" << name << "␣";
   }

Mitarbeiter::~Mitarbeiter()    {
   datei << "-M␣" << name;
   }
```

```
/* Aufgabe a */
/* Implementierung von
void SetGehalt(double g)
void SetGehalt(double t[], int anz);
string GetName();
void SetName(string& n);
void SetName(string* pn);
void SetChef(Mitarbeiter* pc);
void SetChef(Mitarbeiter& c);
*/


/* Aufgabe b */
int SetzeKompliziert1(int teil[]){
   int gehalt = 0;
   int i(0);
   while (teil[i]) {
      gehalt += teil[i];
      ++i;
      datei << gehalt << "␣";
      }
   return gehalt;
   }

int SetzeKompliziert2(int teil[]) {
   int* p= &teil[1];
   int gehalt = 0;
   while (*p) {
      // *(p++); entspricht
      // operator*(operator++(p))
      // Ist es Post- oder Pre-Increment???
      gehalt += *(p++);
      datei << *p << "␣" << gehalt<<"␣--␣";
      }
   return gehalt;
   }

void TestKompliziert() {
   int feld[]={5,4,3,2,1,0};
   int erg = SetzeKompliziert1(feld);
   datei << "\nKompliziert1:␣" << erg <<endl;

   erg = SetzeKompliziert2(feld);
   datei << "\nKompliziert2:␣" << erg <<endl;
   }
```

c.) Implementieren Sie für die Datei `Mitarbeiter.h` einen Include-Wächter.

d.) Welche Methoden der Klasse `Mitarbeiter` können als konstante Methoden implementiert werden? Begründen Sie Ihre Entscheidung!

(*_____*)
Extrablatt benutzen
(*_____*)

Aufgabe 2 : Konstruktoren und Destruktoren

ca. 28 Punkte
Gegeben ist die Klasse `Mitarbeiter` aus der vorherigen Aufgabe und die Klasse `Chef`.

Listing 3: (./Code1/Chef.h)

```
#include "Mitarbeiter.h"
#include <list>

class Chef : public Mitarbeiter
   {
   public:
      typedef Mitarbeiter* MitarbZeiger;
#ifdef ALS_LISTE
      list<MitarbZeiger> p_untergebene;
#else
      // Array von Zeigern auf Mitarbeiter
      MitarbZeiger* p_untergebene;
#endif
      // Eintraegeanzahl im Feld p_untergebene
      int mitarbAnzahl;

   public:
      Chef(double g);
      Chef(string n, double g);
      Chef(const Chef& c2);
      ~Chef();
      Chef& operator=(const Chef& c2);

      virtual double GetGehalt() {return 1000;}

      void AddMitarbeiter(Mitarbeiter* m);
      // true, wenn *m nicht im Feld
      // p_untergebene vorhanden ist.
      bool MitarbeiterNochNichtVorhanden(
         Mitarbeiter* m);
   };
```

Listing 4: (./Code1/Chef.cpp)

```
#include "Chef.h"

Chef::Chef(double g) {
   datei << "+C␣";
   p_untergebene = 0;
   mitarbAnzahl=0;
   gehalt = g;
   }

Chef::Chef(string n, double g){
   datei << "+C␣";
   p_untergebene = 0;
   mitarbAnzahl=0;
   name = n;
   gehalt = g;
   }

Chef::~Chef(){
   datei << "-C␣";
   }
```

a.) Was gibt der Aufruf der folgenden Funktion `FuncAll` in `datei` aus?
Hilfe: Es geht hierbei um die Ausgabe der Konstruktoren und Destruktoren der beteiligten Klassen.
b.) Warum kann von der Klasse `Chef` kein Feld angelegt werden, z.B. `Chef feld[20]`?
Ist aber das folgende Zeigerfeld ohne Erweiterung der Klasse `Chef` möglich `Chef* feld[3]`;? Wenn nein, wie muss die Klasse erweitert werden, damit es trotzdem möglich ist?

Listing 5: (./Code1/MitChef.cpp)

```
#include "Chef.h"
#include "MitChef.h"//nur zur Korrektur wichtig

//Ausgabe 2a
void Func1() {
   datei << "\nFunc1␣";
   Mitarbeiter fritz(14, "Fritz");
   Chef hans(46);
   }
void Func2() {
   datei << "\nFunc2␣";
   Mitarbeiter** feld[5];
   datei << "␣Func2␣";
   feld[1]=0;
   }
void Func3() {
   datei << "\nFunc3␣";
   Mitarbeiter* pfritz;
   pfritz = new Mitarbeiter(14, "Fritz");
   Mitarbeiter* phans=0;
   pfritz = phans;
   datei << "\nFunc3␣";
   }
void FuncAll() {
   Func1();
   Func2();
   Func3();
   }
```

c.) Implementieren Sie den Ausgabeoperator von `Chef`, sodass sich die folgende Ausgabe beim Aufruf der Funktion `FillChef` durch den Ausgabeoperator ergibt.
Gehen Sie davon aus, dass der Ausgabeoperator der Basisklasse schon implementiert ist und benutzen Sie ihn daher.

```
Chef Boss: Boss:133
  Fritz:16
  Karl:17
  Heiner:18

m1: Fritz:16
```

Listing 6: (./Code1/MitChef4.cpp)

```
#include "Chef.h"
#include "MitChef.h"//nur zur Korrektur wichtig

//Aufgabe 2c
void Chef::AddMitarbeiter(Mitarbeiter* m) {
   if (MitarbeiterNochNichtVorhanden(m)) {
      ++mitarbAnzahl;
      MitarbZeiger* hlp =
         new MitarbZeiger [mitarbAnzahl];
      for (int i=0;i<mitarbAnzahl-1; ++i) {
         hlp[i] = p_untergebene[i];
         }
      delete [] p_untergebene;
      p_untergebene = hlp;
      p_untergebene[mitarbAnzahl-1] = m;
      m->SetChef(this);
      }
   }

bool operator==(Mitarbeiter&m1, Mitarbeiter&m2){
   return m1.GetName() == m2.GetName();
   }

bool Chef::MitarbeiterNochNichtVorhanden
                        (Mitarbeiter* m) {
   for (int i=0;i<mitarbAnzahl; ++i) {
      if (*p_untergebene[i] == *m) {
         return false;
         }
      }
   return true;
   }

void FillChef() {
   datei << "\n\nFillChef:" << endl;
   Mitarbeiter m1(16, "Fritz");
   Mitarbeiter m2(17, "Karl");
   Mitarbeiter m3(18, "Heiner");
   Chef ch1("Boss", 133);
   ch1.AddMitarbeiter(&m1);
   ch1.AddMitarbeiter(&m2);
   ch1.AddMitarbeiter(&m3);
   datei << "\nChef␣Boss:␣" << ch1 << endl;
   datei << "m1:␣" << m1 << endl;
   }
```

d.) Was kann im Destruktor der Klasse `Chef` verbessert werden? (Hinweis: Lösung ergibt sich aus der Methode `Chef::AddMitarbeiter`.)
e.) Implementieren Sie den Copy-Konstruktor der Klasse `Chef`. Sie dürfen davon ausgehen, dass der Copy-Konstruktor der Basisklasse `Mitarbeiter` bereits korrekt implementiert wurde.

Aufgabe 3 : STL

ca. 11 Punkte
a.) Wie ändert sich die Implementierung der Methode `Chef::AddMitarbeiter(Mitarbeiter* m)`, wenn

```
MitarbZeiger* p_untergebene;
```

nicht als Zeigerfeld, sondern als STL-Liste implementiert wird:

```
list<MitarbZeiger> p_untergebene;
```

b.) Wie ändert sich entsprechend die Implementierung der Methode `Chef::MitarbeiterNochNichtVorhanden`, wenn `MitarbZeiger* p_untergebene;` nicht als Zeigerfeld, sondern als STL-Liste implementiert wird:

```
list<MitarbZeiger> p_untergebene;
```

? Verwenden Sie zur Implementierung hiervon den STL-Algorithmus `find`, wobei der folgende Vergleichsoperator für die Klasse `Chef` als vorhanden vorausgesetzt werden darf:

```
bool operator==(const Mitarbeiter* m1,
                const Mitarbeiter& m2){
   return m1->GetName() == m2.GetName();
   }
```

Aufgabe 4 : Polymorphie

ca. 9 Punkte
Was gibt der Aufruf der folgenden Funktion `Polymorphie` allein beim Durchlaufen der `if`-Anweisung in `datei` aus? Die Ausgabe der ersten drei Anweisungen soll somit vernachlässigt werden.

Listing 7: (./Code1/MitChef3.cpp)

```
#include "Chef.h"
#include "MitChef.h"//nur zur Korrektur wichtig

void CallFunc(Mitarbeiter* m) {
   datei <<"m␣␣:" << m->GetGehalt() <<endl;
   }
void CallFuncValue(Mitarbeiter m) {
   datei <<"m␣␣:" << m.GetGehalt() <<endl;
   }
void CallFuncRef(Mitarbeiter& m) {
   datei <<"\nm␣␣:" << m.GetGehalt() <<endl;
   }
void CallFuncConst(const Mitarbeiter& m) {
   Mitarbeiter& m2 = const_cast<Mitarbeiter&>(m);
   datei <<"m␣␣:" << m2.GetGehalt() <<endl;
   }

void Polymorphie() {
   datei << "\n\nPolymorphie:" << endl;
   Mitarbeiter m1(16, "Fritz");
   Chef ch1("Boss", 133);
   if (m1.GetGehalt() > -4999999) {
      // Ausgabe ab hier angeben
      datei << "\nAusgabe␣ab␣hier\n";
      datei<< "m1␣:" <<  m1.GetGehalt() <<endl;
      datei <<"ch1:" << ch1.GetGehalt() <<endl;
      CallFunc(  &  ch1);
      CallFuncValue(ch1);
      CallFuncRef(  ch1);
      CallFuncConst(ch1);
      datei << "\nAusgabe␣bis␣hier";
      }
   }
```

Beachten Sie die Implementierung der Methoden `GetGehalt` in Listing 1 und Listing 3.

Aufgabe 5 : Saubere Programmierung

ca. 15 Punkte

a.) Was gibt die Funktion `SchlechterStil` beim Aufruf mit Parameter 134 in `datei` aus?

b.) Der folgende Code ist zwar syntaktisch richtig. Was kann aber vom Standpunkt des Software Engineerings, der defensiven sauberen Programmierung bzw. der Wartbarkeit verbessert werden? Beziehen Sie sich dabei jeweils auf die angegebenen Zeilennummern. Geben Sie zur Lösung der Aufgabe Ihren Verbesserungsvorschlag an.

Hinweis: Es können mehr als 13 Dinge verbessert werden.

Listing 8: (./Code1/SchlechterStil.cpp)

```
#include "Chef.h"                          /* 0 */

SchlechterStil(int param) {                /* 1 */
   float PI = 3.14;                        /* 2 */
   double* radius = new double(10);
   double umfang[2] = {0, 0};              /* 4 */
   int i(0);                               /* 5 */
   if (PI == 3.14)                         /* 6 */
      umfang[0] = 2 * PI * *radius;
   else                                    /* 8 */
      umfang[1]= 2 * 3.14 * *radius;

   while (i<2) {                           /* 10 */
      datei << (i+1) << ".Umfang:"         /* 11 */
                  << umfang[i++] << "␣";
      }                                    /*13 */
   }                                       /*15 */
```